COMP2521 Sort Detective Lab Report

by Evan Zhang (z5383657) Angela Shan (z5358441)

In this lab, the aim is to measure the performance of two sorting programs, without access to the code, and determine which sorting algorithm each program uses.

Experimental Design

We measured how each program's execution time varied as the size and initial sortedness of the input varied. We used the following kinds of input for the following reasons:

- **Bubble sort** - will perform worst on sorted descending O(n^2), best on sorted ascending O(n), is stable, is adaptive
- **Selection sort** - always O(n^2)
- **Insertion sort** - will perform worst on sorted descending O(n^2), best on sorted ascending O(n), is stable, is adaptive
- **Merge sort** - always O(nlog(n))
- **Naive quicksort** - worst with sorted ascending/descending O(n^2)
- **Median-of-three quicksort** - good when list is sorted ascending/descending
- **Randomised quicksort** - only algorithm where time changes significantly between exact same test
- **Bogo sort** - Average time complexity $O((n+1)!)$ -> long ass time

Because of the way timing works on Unix/Linux, it was necessary to repeat the same test 3 times each.

We also investigated the stability of the sorting programs by testing with identical elements with an extra different indicative element attached.

We also investigated the adaptability of the programs by inputting a sorted list and compared it to how it performed with a random list.

Experimental Results

For Program A, we observed that:

- It preserved order of identical elements in the list -> algorithm is stable
- Spent significantly longer on random and descending sorted
- Spent a relatively short time on ascending sorted -> algorithm could be adaptive
- Tested using ascending sorted with last one in front -> insertion sort perform well, bubble sort perform badly, therefore was insertion sort, since performed well

For Program B, we observed that the program:

- Did not preserve order of identical items -> is unstable

- Takes longer to sort a random list vs sorted ascending/descending list -> cannot be naive quicksort (where asc/desc is worse), cannot be selection or merge sort (where time complexity is same for R, A, D)
- Takes roughly the same amount of time to sort ascending vs descending list (0.01 s diff) -> cannot be bubble or insertion sort
- Time did not change significantly when repeating the same test -> cannot be randomised quicksort

Conclusions

On the basis of our experiments and our analysis above, we believe that

- sortA implements the *insertion* sorting algorithm
- sortB implements the *median-of-three quicksort* sorting algorithm

Appendix

Any large tables of data that you want to present …

Input	*sortA Time (seconds)*	*sortB Time (seconds)*
100000 R	*38.20*	*0.02*
100000 A	*0.01*	*0.02*
100000 D	*35.40*	*0.01*
1000000 R		*0.38*
1000000 A	*0.12*	*0.15*
1000000 D		*0.16*